Coding in-place tooltips

devblogs.microsoft.com/oldnewthing/20060626-11

June 26, 2006

Raymond Chen

Today we'll look at how to implement in-place tooltips. These are tooltips that appear when the user hovers the mouse over a string that cannot be displayed in its entirety. The tooltip overlays the partially-displayed text and provides the remainder of the text that had been truncated. The keys to this technique are the `TTN_SHOW` notification (which lets you adjust the positioning of a tooltip before it is shown) and the `TTM_ADJUSTRECT` message which tells you precisely where you need the tooltip to be.

Start with our scratch program and add the following:

```
HFONT g_hfTT;
HWND g_hwndTT;
RECT g_rcText;
LPCTSTR g_pszText = TEXT("Lorem ipsum dolor sit amet.");
const int c_xText = 50;
const int c_yText = 50;
BOOL
OnCreate(HWND hwnd, LPCREATESTRUCT lpcs)
{
 g_hwndTT = CreateWindowEx(WS_EX_TRANSPARENT, TOOLTIPS_CLASS, NULL,
                           TTS_NOPREFIX,
                           0, 0, 0, 0,
                           hwnd, NULL, g_hinst, NULL);
 if (!g_hwndTT) return FALSE;
 g_hfTT = GetStockFont(ANSI_VAR_FONT);
 SetWindowFont(g_hwndTT, g_hfTT, FALSE);
 HDC hdc = GetDC(hwnd);
 HFONT hfPrev = SelectFont(hdc, g_hfTT);
 SIZE siz;
 GetTextExtentPoint(hdc, g_pszText, lstrlen(g_pszText), &siz);
 SetRect(&g_rcText, c_xText, c_yText,
                    c_xText + siz.cx, c_yText + siz.cy);
 SelectFont(hdc, hfPrev);
 ReleaseDC(hwnd, hdc);
 TOOLINFO ti = { sizeof(ti) };
 ti.uFlags = TTF_TRANSPARENT | TTF_SUBCLASS;
 ti.hwnd = hwnd;
 ti.uId = 0;
 ti.lpszText = const_cast<LPTSTR>(g_pszText);
 ti.rect = g_rcText;
 SendMessage(g_hwndTT, TTM_ADDTOOL, 0, (LPARAM)&ti);
 return TRUE;
}
void
PaintContent(HWND hwnd, PAINTSTRUCT *pps)
{
 HFONT hfPrev = SelectFont(pps->hdc, g_hfTT);
 TextOut(pps->hdc, g_rcText.left, g_rcText.top,
         g_pszText, lstrlen(g_pszText));
 SelectFont(pps->hdc, hfPrev);
}
```

After declaring a few variables, we dig into our computations at window creation. We create the tooltip window, passing ourselves as the owner window. (Passing ourselves as the owner window is important in order to get proper Z-order behavior. I refer the reader to the fifth of my “Five Things Every Win32 Developer Should Know” topics for further details.) We then obtain our font and set it into the tooltip control so that the tooltip renders in the same font we do. (I’ll take up more complex font manipulation in a future entry.) We then measure our text in the target font and set the `g_rcText` rectangle to the dimensions of that text. We use that rectangle to establish the boundaries of a tool in the tooltip control. By setting the

TTF_SUBCLASS flag, we indicate that the tooltip control should subclass our window in order to intercept mouse messages. This is a convenience to avoid us having to use the TTM_RELAYEVENT message to forward the mouse messages manually. This hooks up the tooltip.

Painting the content is a simple matter of selecting the font and drawing the text.

Run this program and hover over the text. The tooltip appears, but it's in the wrong place. Aside from that, though, things are working as expected. The tooltip has the correct font, it fires only when the mouse is over the text itself, and it dismisses when the mouse leaves the text. Let's position the tooltip:

```
LRESULT
OnTooltipShow(HWND hwnd, NMHDR *pnm)
{
 RECT rc = g_rcText;
 MapWindowRect(hwnd, NULL, &rc);
 SendMessage(pnm->hwndFrom, TTM_ADJUSTRECT, TRUE, (LPARAM)&rc);
 SetWindowPos(pnm->hwndFrom, 0, rc.left, rc.top, 0, 0,
   SWP_NOACTIVATE | SWP_NOSIZE | SWP_NOZORDER);
 return TRUE; // suppress default positioning
}
LRESULT
OnNotify(HWND hwnd, int idFrom, NMHDR *pnm)
{
 if (pnm->hwndFrom == g_hwndTT) {
  switch (pnm->code) {
  case TTN_SHOW:
   return OnTooltipShow(hwnd, pnm);
  }
 }
 return 0;
}
// Add to WndProc
    HANDLE_MSG(hwnd, WM_NOTIFY, OnNotify);
```

The TTN_SHOW notification is sent when the tooltip is about to be displayed. We respond to the notification by mapping the text rectangle to screen coordinates and using the TTM_ADJUSTRECT message to expand the rectangle to include all the margins and borders that the tooltip control will place around the text. That way, when we position the tooltip at that location, the margins and borders match up precisely, and the text appears at the desired location. It is important to return TRUE to indicate to the tooltip control that we took care of positioning the window and it should not do its default positioning.

When you run this program, you will find one more problem: Tooltip animations are still taking place, which is particularly distracting if the animation is a slide animation. This is easy to fix: Tweak the way we create the tooltip control.

```
g_hwndTT = CreateWindowEx(WS_EX_TRANSPARENT, TOOLTIPS_CLASS, NULL,
                          TTS_NOPREFIX | TTS_NOANIMATE,
                          0, 0, 0, 0,
                          hwnd, NULL, g_hinst, NULL);
```

The `TTS_NOANIMATE` style suppress animations, which means that the tooltip simply pops into place, exactly what we want.

So there you have it—the basics of in-place tooltips. Of course, there are many details you may wish to deal with, such as showing the tooltip only if the string is clipped. But those issues are independent of in-place tooltips, so I won't go into them here. We'll look at selected aspects of tooltips in future installments.

Raymond Chen

Follow